[79] A. Homescu, S. Neisius, P. Larsen, S. Brunthaler, and M. Franz, “Profile-guided Automated Software Diversity,” in *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2013, Shenzhen, China, February 23-27, 2013*, pp. 23:1–23:11, IEEE Computer Society, 2013.

[80] S. Bhatkar, D. C. DuVarney, and R. Sekar, “Address Obfuscation: An Efficient Approach to Combat a Board Range of Memory Error Exploits,” in *Proceedings of the USENIX Security Symposium*, 2003.

[81] S. Bhatkar and D. C. DuVarney, “Efficient Techniques for Comprehensive Protection from Memory Error Exploits,” in *Proceedings of the 14th USENIX*, 2005.

[82] K. Pettis and R. C. Hansen, “Profile Guided Code Positioning,” in *Proceedings of the ACM SIGPLAN’90 Conference on Programming Language Design and Implementation (PLDI)*, pp. 16–27, 1990.

[83] S. Crane, A. Homescu, S. Brunthaler, P. Larsen, and M. Franz, “Thwarting Cache Side-channel Attacks Through Dynamic Software Diversity,” in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*, The Internet Society, 2015.

[84] A. Romano, D. Lehmann, M. Pradel, and W. Wang, “Wobfuscator: Obfuscating JavaScript Malware via Opportunistic Translation to WebAssembly,” in *2022 2022 IEEE Symposium on Security and Privacy (SP) (SP)*, (Los Alamitos, CA, USA), pp. 1101–1116, IEEE Computer Society, may 2022.

[85] M. T. Aga and T. M. Austin, “Smokestack: Thwarting DOP Attacks with Runtime Stack Layout Randomization,” in *IEEE/ACM International Symposium on Code Generation and Optimization, CGO*, pp. 26–36, 2019.

[86] S. Lee, H. Kang, J. Jang, and B. B. Kang, “SaVioR: Thwarting Stack-based Memory Safety Violations by Randomizing Stack Layout,” *IEEE Trans. Dependable Secur. Comput.*, vol. 19, no. 4, pp. 2559–2575, 2022.

[87] Y. Younan, D. Pozza, F. Piessens, and W. Joosen, “Extended Protection against Stack Smashing Attacks without Performance Loss,” in *22nd Annual Computer Security Applications Conference (ACSAC 2006), 11-15 December 2006, Miami Beach, Florida, USA*, pp. 429–438, IEEE Computer Society, 2006.

[88] Y. Xu, Y. Solihin, and X. Shen, “MERR: Improving Security of Persistent Memory Objects via Efficient Memory Exposure Reduction and Randomization,” in *ASPLOS ’20: Architectural Support for Programming Languages and Operating Systems*, pp. 987–1000, 2020.

[89] G. S. Kc, A. D. Keromytis, and V. Prevelakis, “Countering Code-injection Attacks With Instruction-set Randomization,” in *Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS*, pp. 272–280, 2003.

[90] E. G. Barrantes, D. H. Ackley, T. S. Palmer, D. Stefanovic, and D. D. Zovi, “Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks,” in *Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS*, pp. 281–289, 2003.

[91] M. Chew and D. Song, “Mitigating Buffer Overflows by Operating System Randomization,” Tech. Rep. CS-02-197, Carnegie Mellon University, 2002.

[92] D. Couroussé, T. Barry, B. Robisson, P. Jaillon, O. Potin, and J. Lanet, “Runtime Code Polymorphism as a Protection Against Side Channel Attacks,” in *Proceedings of Information Security Theory and Practice - 10th IFIP WG 11.2 International Conference, WISTP*, vol. 9895, pp. 136–152, 2016.

[93] M. Jacob, M. H. Jakubowski, P. Naldurg, C. W. Saw, and R. Venkatesan, “The Superdiversifier: Peephole Individualization for Software Protection,” in *Proceedings of Advances in Information and Computer Security, Third International Workshop on Security, IWSEC 2008*, vol. 5312, pp. 100–120, 2008.

[94] M. Henry, “Superoptimizer: A Look at the Smallest Program,” *ACM SIGARCH Computer Architecture News*, vol. 15, pp. 122–126, Nov 1987.

[95] V. Le, M. Afshari, and Z. Su, “Compiler Validation via Equivalence Modulo Inputs,” in *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI*, pp. 216–226, 2014.

[96] B. R. Churchill, O. Padon, R. Sharma, and A. Aiken, “Semantic Program Alignment for Equivalence Checking,” in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI*, pp. 1027–1040, 2019.

[97] V. Le, M. Afshari, and Z. Su, “Compiler Validation via Equivalence Modulo Inputs,” in *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI*, pp. 216–226, 2014.

[98] N. Harrand, C. Soto-Valero, M. Monperrus, and B. Baudry, “Java Decompiler Diversity and Its Application to Meta-decompilation,” *J. Syst. Softw.*, vol. 168, p. 110645, 2020.

[99] M. Zalewski, “American Fuzzy Lop,” 2017.